Analysing ZigBee Key Establishment Protocols

Ender Yüksel

ey@imm.dtu.dk

26 February 2010

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk

Preface

This report provides a detailed documentation on the application of static program analysis to the key establishment protocols of the ZigBee wireless sensor network standard. The approach presented in this report is within the scope of the SENSORIA (*Software Engineering for Service-Oriented Overlay Computers*) project, and will form a preliminary version of one of the chapters in my PhD dissertation. The discovered flaw and the proposed secure protocols were recently published in a conference paper (see references [17]) and also accepted for journal publication.

Acknowledgement This work has been partially supported by EU-FETPI Global Computing Project IST-2005-16004 SENSORIA.

I would like to thank Hanne Riis Nielson and Flemming Nielson for the supervision and invaluable contribution in this work, and also for providing me with this opportunity to work on an international research project.

I would also like to thank Gavin Lowe for kind discussions and feedback on fixing the flaw in the key establishment protocol.

Kongens Lyngby, February 2010

Ender Yüksel

Contents

List of Tables

Analyzing the Protocols

Computer networks or simply networks are the main means of information sharing and communication in today's IT infrastructure. Certain protocols are executed to facilitate communication in networks. However, such networks are mostly insecure and the communication needs to be protected against attackers that may influence network traffic and and communication parties that might be either dishonest or compromised by attackers.

Cryptographic security protocols form an essential ingredient of network communications by ensuring secure communication over insecure networks. These protocols use cryptographic primitives to support certain security properties, but ensuring this properties requires a lot more effort. Despite the relatively small size of the security protocols it is very hard to design them correctly, and their analysis is very complicated. One of the most well-known examples is the Needham-Schroeder protocol [1], that was proven secure by using BAN logic [2]. Seventeen years later G. Lowe [3, 4], found a flaw by using an automatic tool FDR. The flaw was not detected in the original proof because of different assumptions on the intruder model. The fact that this new attack had escaped the attention of the experts was an indication of the underestimation of the complexity of protocol analysis. This example has shown that protocol analysis is critical for assessing the security of such cryptographic protocols.

In this report, we present our approach for protocol analysis together with a real example where we find an important flow in a contemporary wireless sensor network security protocol. We start by modelling protocols using a specific process algebraic formalism called LySA process calculus. We then apply an analysis based on a special program analysis technique called control flow anal-

ysis. We apply this technique to the ZigBee-2007 End-to-End Application Key Establishment Protocol and with the help of the analysis discover an unknown flaw. Finally we suggest a fix for the protocol, and verify that the fix works by using the same technique.

1.1 An Overview of the Analysis Method

Static program analysis, in essence, examines a program statically, before any attempt of execution. Although the finite amount of resources may limit the information or the answers to important questions, the approximation based approach of static program analysis makes it preferable on the area of protocol analysis. Instead of facing undecidability problem, this technique sacrifices precision and gives approximate answers about a property of a certain program, or a piece of code, or a protocol as in our case. However, this loss of precision does not mean that we are missing the flaws, it merely means that the analysis results may include false positives, such as a bug or a flaw that the program does not contain.

Static program analysis was originally developed for generating codes and optimising compilers [5, 6]. Nevertheless, the analysis technique have recently been directed to the field of security. Encouraging results have been obtained by the use of this approach where safe approximations to the set of values or behaviours arising during protocol runs can be predicted.

Control flow analysis of processes formalised in the LySa process calculus successfully computes an over-approximation of the run-time behaviour of a protocol [7, 8]. This method is actually the protocol analysis method that we present in this report. The roadmap of the analysis method is given in Fig. 1.1, and we will present the steps of this roadmap in the following sections.

1.2 Modelling in LySa Process Calculus

The first step in the protocol analysis is to formalise the protocol narration into a model that is suitable for the analysis. In our case, we formalize the protocols using the LySa process calculus [8]. LySa is based on the π-calculus [9] and incorporates cryptographic operations using ideas from the Spi-calculus [10]. However, LySa has two different properties compared to spi/π calculus. First, LySa has one global ether, instead of channels. The reason for this difference is

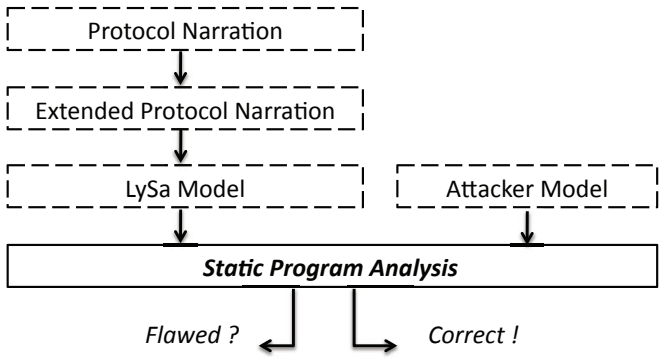

Figure 1.1: The Roadmap of the Analysis

Table 1.1: LySA Terms - Symmetric Fragment

$E ::=$		
	x	variable
$\mid$	n	name
$\mid$	$\{E_1, \ldots, E_k\}^{\ell}_{E_0}[\mathsf{dest}\,\mathcal{L}]$	symmetric encryption

that, in usual networking implementations (e.g. ethernet-based, wireless, etc.), anyone can eavesdrop or act as an active attacker which does not correspond to the channel-based communication. The second difference is in the pattern matching usage in the tests of the expressions associated with input and decryption. Although LySA is a very powerful process calculus which also supports asymmetric encryption, digital signatures, etc., in order to make it simple we only illustrate the *symmetric* fragment. The symmetric fragment suffices to prove our claims in the example that we will present the flaw discovery since the protocol is designed for symmetric encryption only. The reader interested in further details including the asymmetric fragment may refer to [8].

In LySA, we have terms (E) that consist of names (keys, nonces, messages, etc.), variables, and the compositions of them using symmetric encryption. The syntax of terms is shown in Table 1.1. In the case of encryption, the tuples of terms $E_1, \ldots, E_k$ are encrypted under a term E_0 which actually represents an encryption key. Note that an assumption of perfect cryptography is adopted, which means that *decryption with the correct key* is the only inverse function of encryption. The *annotation* inside brackets in the end of encryption will be explained later in this section.

The syntax of the processes (P) which is mostly alike to the polyadic Spi-calculus [10] is shown in Table 1.2. At this point, we prefer to skip the syntax

Table 1.2: LySA Processes - Symmetric Fragment

$P ::=$		
	0	nil
|	$P_1 \mid P_2$	parallel comp.
|	$!P$	replication
|	$(\nu\, n)\, P$	restriction
|	$\langle E_1, \ldots, E_k \rangle.P$	output
|	$(E_1, \ldots, E_j; x_{j+1}, \ldots, x_k).P$	input
|	$\mathsf{decrypt}\ E\ \mathsf{as}\ \{E_1, \ldots, E_j;\ x_{j+1}, \ldots, x_k\}^{\ell}_{E_0} [\mathsf{orig}\, \mathcal{L}\,]\ \mathsf{in}\ P$	symm. decryption

for simple ones in the table, but explain the more interested and complicated two: output and input processes. The output process $\langle E_1, \ldots, E_k \rangle.P$ sends the k-tuple $E_1, \ldots, E_k$ to the network and continues as process P. Similarly, the input process $(E_1, \ldots, E_j; x_{j+1}, \ldots, x_k).P$ receives a k-tuple $E'_1, \ldots, E'_k$ and if conditions are satisfied, removes the k-tuple from the network. Here, the input operation uses pattern matching which will only succeed if the prefix of the input message matches the terms specified before the semi-colon. In a simple manner, we can say that for some input E' the input process $(E;x).P$ means that if E' can be separated into two parts such that first part pairwise matches to the values E, then the remaining part of the input will be bound to the variables x. As you can see in Table 1.2, the number of tuples in E' is k so that this is the total number of tuples in E and x. This kind of pattern matching is also used in decryption.

Example 1.a *The example* LySA *code below is a* ***new*** *(created - restriction) encryption key (K) followed by an* ***output*** *which includes three plaintext elements (A, B, K_A) and an encrypted element ($\{K\}_{K_A}$).*

$$(\nu\, K)\, \langle A, B, K_A, \{K\}_{K_A} \rangle$$

Example 1.b *The example* LySA *code below is an* ***input*** *that binds the last two elements of the input to the variables x_{KA} and x as long as the first two elements are A and B.*

$$(A, B; x_{KA}, x)$$

Example 1.c *The example* LySA *code below is a* ***decryption*** *that decrypts the value bound to variable x using the encryption key bound to variable x_{KA} and binds the resulting plaintext value to the variable x_K. Note that this decryption always succeeds without any need of pattern matching, as long as the correct key exists in the receiver.*

$$\mathsf{decrypt}\ x\ \mathsf{as}\ \{;\, x_K\}_{x_{KA}}$$

In order to describe the *message authentication* intentions of the protocols, we also have *annotation*s for origin and destination. Encryptions can be annotated with fixed labels called *crypto-point*s that define their positions in the process, and with *assertion*s that specify the origin and destination of encrypted messages. A crypto-point ℓ is an element of some set $\mathcal{C}$ and used when encryptions/decryptions occur. The LySA term for encryption:

$$\{E_1, \ldots, E_k\}^{\ell}_{E_0}[\mathsf{dest}\,\mathcal{L}\,]$$

means that the encryption happened at crypto-point ℓ and the assertion [dest $\mathcal{L}$] means that corresponding (valid) decryption is *to* happen at a crypto-point that belongs to the set $\mathcal{L}$ such that $\mathcal{L} \subseteq \mathcal{C}$. Similarly, in the LySA term for decryption:

$$\mathsf{decrypt}\ E\ \mathsf{as}\ \{E_1, \ldots, E_j;\, x_{j+1}, \ldots, x_k\}^{\ell}_{E_0}[\mathsf{orig}\,\mathcal{L}\,]\ \mathsf{in}\ P$$

[orig $\mathcal{L}$] specifies the crypto-points $\mathcal{L} \subseteq \mathcal{C}$ that E is allowed to have been encrypted.

Example 2 *The example* LySA *code below is the* ***composition*** *of the three separate parts in Example 1, and the necessary* ***annotations*** *in such a way that now we have two separate processes running in* ***parallel****.*

$$\begin{array}{ll} /*\ \text{a}\ */ & (\nu\, K)\,\langle A, B, K_A, \{K\}^{\ell_A}_{K_A}[\mathsf{dest}\,\{\ell_{\mathcal{B}}\}\,]\rangle.0 \\ & \mid \\ /*\ \text{b}\ */ & (A, B; x_{KA}, x). \\ /*\ \text{c}\ */ & \mathsf{decrypt}\ x\ \mathsf{as}\ \{;\, x_K\}^{\ell_B}_{x_{KA}}[\mathsf{orig}\,\{\ell_{\mathcal{A}}\}\,]\ \mathsf{in}\ .0 \end{array}$$

The example we constructed step by step is actually the LySA *model of the single-message protocol below:*

1. A → B: KA, $\{\mathrm{K}\}_{KA}$

The upper part (line a) of the parallel composition is the code for principal A, and the lower part (lines b and c) is for principal B. In this example, annotations

Table 1.3: Extended Protocol Narration - Case 1

1. A	$\rightarrow$		:A, TC, {TC, AppKey, B}$_{KA}$
			[dest TC]
1'.	$\rightarrow$	**TC**	:$x_{initiator}$, x_{TC}, $x_{message}$
			[check x_{TC}=TC]
1".		**TC**	:decrypt $x_{message}$ as
			$\{x'_{TC}, x_{keytype}, x_{responder}\}_{KA}$
			[orig x_A][check x'_{TC}=TC, $x_{keytype}$=AppKey]
2. TC	$\rightarrow$		:[new LK]
			TC, $x_{initiator}$,
			$\{x_{initiator}$,AppLK,$x_{responder}$,TRUE,LK$\}_{KA}$
			[dest $x_{initiator}$]
2'.	$\rightarrow$	**A**	:y_{TC}, y_A, $y_{message}$
			[check y_{TC}=TC, y_A=A]
2".		**A**	:decrypt $y_{message}$ as
			$\{y'_A, y_{keytype}, y_B, y_{bool}, y_{LK}\}_{KA}$
			[orig TC][check y'_A=A, $y_{keytype}$=AppLK]
			[check y_B=B, y_{bool}=TRUE]

state that the encryption at crypto-point ℓ_A is intended to be decrypted only at ℓ_B. In a corresponding manner, the decryption at ℓ_B should originate from the encryption at ℓ_A.

1.2.1 Specifying Protocols in LySa

In the beginning, we have a protocol narration like the one in Table 1.7. Then we extend the narration to specify the internal actions to be performed in principals when receiving those messages. The reason for this kind of extension or conversion is to completely state the actions internal to the principals, which are normally left implicit in the narration of security protocols.

As an example, the extended protocol narration of (due to the lack of space) the first two messages of Case 1 is given in Table 1.3. For each message in the original protocol narration, we have an output message n and an input

message n' in the extended protocol narration. Input message n' presents the variable (those written in *italics*) bindings and necessary checks in the receiver side. If a variable is a ciphertext and the receiver has the correct encryption key, then we have another message (i.e. n'') for each of those variables. In addition, we explicitly write the internal actions as annotations between square brackets, in order to bridge the gap between informal and formal specification of the protocol. Note that when analysing protocols we add an extra message to the end, where a principal attempts to communicate the other through the new shared key, LK. For example, the message

$$\mathbf{1.\ B} \rightarrow \mathbf{A:}\ \{\mathrm{MSG}\}_{LK}$$

does not change the protocol nor bring any (nor bring any additional cost to the implementations), it is just a sample message that will be sent using the new LK and thus it will ease the validation which is done by checking attackers knowledge.

In the next phase, we convert the extended protocol narration into a LySA model. We use the LySA syntax that we explained earlier in this section and configure the necessary settings. As an example, a regular LySA model of the protocol that we have used to demonstrate extended protocol conversion is given in Table 1.4. Further details of specifying protocols in LySA are present in [8].

1.3 Static Program Analysis

Static Analysis is a formal method that enables the security analysis of cryptographic communication protocols which are modelled as LySA processes. Messages communicated on the network are tracked with the possible values of the variables in the protocol. Besides, the potential violations of the destination/origin annotations are also recorded. The aim of static analysis is to efficiently compute the safe approximations to the behaviour of the models without actually running them. In Fig. 1.2 we can see the approximation approach. In general, it is impossible to compute the precise answer so we make a choice between over-approximation and under-approximation. Static analysis over-approximates the set of possible operations that the LySA process describes. The nature of over-approximation may cause the analysis to investigate a trace which is impossible at all. However, over-approximation is needed to make a safe approximation since under-approximation could miss some traces.

Table 1.4: LySa Model - Case 1

	let $X \subseteq \mathbf{N}$ s.t. $\lfloor \mathbf{N} \rfloor = \{1,2,3\}$ in
	$(\nu_{i \in X}\ KA_i)\ (\nu_{j \in X}\ KB_j)$
	$\mid_{i \in X} \mid_{j \in X \cup \{0\}}$!
1	$\langle A_i, TC, \{TC, AppKey, B_j\}_{KA_i}$[at $a1_{ij}$ dest $\{tc1_{ij}\}$]$\rangle$.
2'	$(TC, A_i;\ y_{ij})$.
2"	decrypt y_{ij} as $\{A_i, AppLK, B_j, TRUE;\ xLK_{ij}\}_{KA_i}$
	[at $a2_{ij}$ orig $\{tc2_{ij}\}$] in
4'	$(B_j, A_i;\ y2_{ij})$.
4"	decrypt $y2_{ij}$ as $\{;\ xmsg_{ij}\}_{xLK_{ij}}$[at $a4_{ij}$ orig $\{b4_{ij}\}$] in 0
	|
	$\mid_{j \in X} \mid_{i \in X \cup \{0\}}$!
3'	$(TC, B_j;\ z_{ij})$.
3"	decrypt z_{ij} as $\{B_j, AppLK, A_i, FALSE;\ yLK_{ij}\}_{KB_j}$
	[at $b3_{ij}$ orig $\{tc3_{ij}\}$] in
4	$(\nu\ MSG_{ij})\ \langle B_j, A_i, \{MSG_{ij}\}_{yLK_{ij}}$[at $b4_{ij}$ dest $\{a4_{ij}\}$]$\rangle$. 0
	|
	$\mid_{i \in X \cup \{0\}} \mid_{j \in X \cup \{0\}}$!
1'	$(A_i, TC;\ x_{ij})$.
1"	decrypt x_{ij} as $\{TC, AppKey, B_j;\ \}_{KA_i}$
	[at $tc1_{ij}$ orig $\{a1_{ij}\}$] in
2	$(\nu\ LK_{ij})\ \langle TC, A_i, \{A_i, AppLK, B_j, TRUE, LK_{ij}\}_{KA_i}$
	[at $tc2_{ij}$ dest $\{a2_{ij}\}$]$\rangle$.
3	$\langle TC, B_j, \{B_j, AppLK, A_i, FALSE, LK_{ij}\}_{KB_j}$
	[at $tc3_{ij}$ dest $\{b3_{ij}\}$]$\rangle$. 0

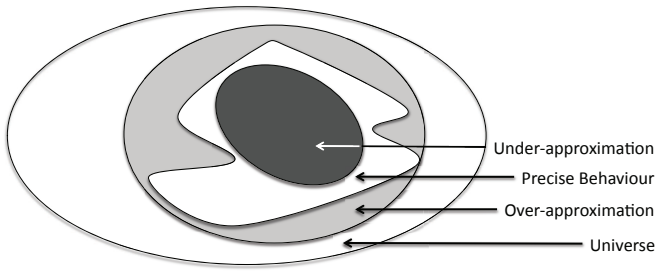

Figure 1.2: Static Analysis

1.3.1 Analysis Method

The static analysis we use in this study is specified as a Flow Logic [7, 8], which is based on the control flow analysis and the data flow analysis techniques that allow us to make it fully automatic [11].

Control flow analysis is a program analysis technique that is used to compute approximations of the result of a program execution without running the program. Such an analysis helps us in determining the sets of values that may be generated by communication using a specific protocol, which is beneficial for validating certain security properties. Especially when used in conjunction with a model of possible malicious activity (i.e. attacker), the analysis provides a safe approximation of all events that may happen.

Flow Logic is a notational style for specifying analyses across programming paradigms, introduced by Nielson, Nielson [12, 13, 14], and with Hankin [11]. By abstracting from domain specific formalisms and instead using standard mathematical notations, the Flow Logic constitutes a meta-language that can present an analysis without requiring additional knowledge about particular formalisms. Deriving an analysis estimate from the resulting analysis specification is then left as a separate activity, usually involving orthogonal considerations and tools. This approach allows the designer to focus on the specification of analyses without making compromises dictated by implementation considerations. Similarly, implementation is simplified and improved, as the implementer is always free to choose the best available tool. In the next sections, we will present control flow analysis of LySA in the style of flow logic.

The control flow analysis that we use in protocol analysis is specified using the flow logic framework as a predicate

$$\rho, \kappa, \psi \models P$$

that holds precisely when ρ, κ, and ψ form an analysis result that correctly describes the behaviour of the process P.

The main components of the analysis are:

- *The variable environment ρ,* an over-approximation of the potential values of each variable that it may be bound to.
- *The network component κ,* an over-approximation of the set of messages that can be communicated over the network.

Table 1.5: Analysis for Terms, $\rho \models E{:}\vartheta$

(AName)	$\dfrac{\lfloor n \rfloor \in \vartheta}{\rho \models n{:}\vartheta}$
(AVar)	$\dfrac{\rho(\lfloor x \rfloor) \subseteq \vartheta}{\rho \models x{:}\vartheta}$
(AEnc)	$\dfrac{\wedge_{i=0}^{k} \rho \models E_i{:}\vartheta_i \quad \wedge \quad \forall V_0, V_1, \ldots, V_k{:} \wedge_{i=0}^{k} V_i \in \vartheta_i \quad \Rightarrow \quad \{V_1, \ldots, V_k\}^{\ell}_{V_0}[\mathsf{dest}\,\mathcal{L}\,] \in \vartheta}{\rho \models \{E_1, \ldots, E_k\}^{\ell}_{E_0}[\mathsf{dest}\,\mathcal{L}\,]{:}\vartheta}$

- *The error component* ψ, the set of error messages in the form (ℓ, ℓ'), indicating that something encrypted at ℓ was unexpectedly decrypted at ℓ'.

The analysis is judgments of the form $\rho, \kappa, \psi \models P$ which express that ρ, κ, ψ compose a valid analysis for the process P. We also need to introduce the auxiliary judgment $\rho \models E{:}\vartheta$ at this point. This expresses that ϑ, the set of values, is an acceptable estimate of the values that the term E may evaluate in ρ, the abstract environment.

To keep the analysis component finite, we partition all the names that are generated by a LySA process into finitely many equivalence classes. A *canonical value* is a representative for each of these equivalence classes. Names from the same equivalence class are assigned a common *canonical name* and instead of the actual names, we use the names of those equivalence classes. For example, the canonical representative of a name n is denoted by $\lfloor n \rfloor$. Since it allows us to analyse an infinite number of principals, canonical value is an important analysis element [15].

The analysis of terms is listed in Table 1.5. The rule for analysing names (AName) states that ϑ is an acceptable estimate for a name n if the canonical representative of n belongs to ϑ. The rule for analysing variables (AVar) states that ϑ is an acceptable estimate for a variable x if it is a superset of $\rho(\lfloor x \rfloor)$. The rule for analysing symmetric encryption (AEnc) finds the set ϑ_i for each term E_i, collects all k-tuples of values $(V_0, \ldots, V_k)$ taken from $\vartheta_0 \times \ldots \times \vartheta_k$ into values of the form $\{V_1, \ldots, V_k\}^{\ell}_{V_0}[\mathsf{dest}\,\mathcal{L}\,]$ and requires that these values belong to ϑ.

The analysis of processes is listed in Table 1.6. The idea of the analysis is very similar to the analysis of terms, therefore instead of explaining all the rules we explain only one interesting rule. The rule for analysing output (AOut) uses the

Table 1.6: Analysis for Processes, $(\rho, \kappa) \models P{:}\psi$

(ANil)	$(\rho, \kappa) \models 0{:}\psi$
(APar)	$\dfrac{(\rho, \kappa) \models P_1{:}\psi \quad \wedge \quad (\rho, \kappa) \models P_2{:}\psi}{(\rho, \kappa) \models P_1 \mid P_2{:}\psi}$
(ARep)	$\dfrac{(\rho, \kappa) \models P{:}\psi}{(\rho, \kappa) \models !P{:}\psi}$
(ANew)	$\dfrac{(\rho, \kappa) \models P{:}\psi}{(\rho, \kappa) \models (\nu\, n)\, P{:}\psi}$
(AOut)	$\dfrac{\begin{array}{c}\wedge_{i=1}^{k} \rho \models E_i{:}\vartheta_i \quad \wedge \\ (\rho, \kappa) \models P{:}\psi \\ \forall V_1, \ldots, V_k{:}\, \wedge_{i=1}^{k} V_i \in \vartheta_i \quad \Rightarrow \quad \langle V_1, \ldots, V_k \rangle \in \kappa \quad \wedge\end{array}}{(\rho, \kappa) \models \langle E_1, \ldots, E_k \rangle.P{:}\psi}$
(AIn)	$\dfrac{\begin{array}{c}\wedge_{j=1}^{k} \rho \models E_i{:}\vartheta_i \quad \wedge \\ (\rho, \kappa) \models P{:}\psi \\ \forall V_1, \ldots, V_k \in \kappa{:}\, \wedge_{j=1}^{k} V_i \in \vartheta_i \quad \Rightarrow \quad \wedge_{i=j+1}^{k} V_i \in \rho(\lfloor x_i \rfloor) \quad \wedge\end{array}}{(\rho, \kappa) \models (E_1, \ldots, E_j;\, x_{j+1}, \ldots, x_k).P{:}\psi}$
(ADec)	$\dfrac{\begin{array}{c}\rho \models E{:}\vartheta \quad \wedge \\ \forall\, \wedge_{i=0}^{j} \rho \models E_i{:}\vartheta_i \quad \wedge \\ (\rho, \kappa) \models P{:}\psi \\ ((\ell \notin \mathcal{L}' \vee \ell' \notin \mathcal{L}) \Rightarrow (\ell, \ell') \in \psi) \quad \wedge \\ \forall \{V_1, \ldots, V_k\}_{V_0}^{\ell}[\mathsf{dest}\, \mathcal{L}\,] \in \vartheta{:}\, \wedge_{i=0}^{j} V_i \in \vartheta_i \quad \Rightarrow \quad \wedge_{i=j+1}^{k} V_i \in \rho(\lfloor x_i \rfloor) \quad \wedge\end{array}}{(\rho, \kappa) \models \mathsf{decrypt}\ E\ \mathsf{as}\ \{E_1, \ldots, E_j;\, x_{j+1}, \ldots, x_k\}_{E_0}^{\ell'}[\mathsf{orig}\, \mathcal{L}\,]\, \mathsf{in}\, P{:}\psi}$

analysis for terms to find the estimate ϑ_i for each term E_i and requires that all all k-tuples of values $\langle V_1, \ldots, V_k \rangle$ taken from $\vartheta_1 \times \ldots \times \vartheta_k$ are in κ (i.e. they may flow on the network). The rule also requires that the components ρ, κ, ψ compose a valid analysis for process P.

Example 3 *Static analysis of the* LySA *model given in Example 2 will lead to the following results:*

$\langle A, B, K_A, \{K\}^{\ell_A}_{K_A}[\mathsf{dest}\,\ell_{\mathcal{B}}]\rangle \in \kappa$
$K_A \in \rho(x_{KA})$
$\{K\}^{\ell_A}_{K_A}[\mathsf{dest}\,\ell_{\mathcal{B}}] \in \rho(x)$
$K \in \rho(x_K)$

Looking at the results above, it is easy to see that the first line is related to ***line a*** *in Example 2. Likewise, next two lines derived from* ***line b*** *and the last line derived from* ***line c*** *in Example 2. Note that, how the analysis works is not the subject of this paper. Therefore, see [8] for how Example 2 leads to Example 3.*

1.3.2 Attacker Model

In practice, network protocols are vulnerable to attacks. Unfortunately it is even easier to attack wireless networks since any computer within range that is equipped with a wireless client card can pull the signal and access the data. In this study, LySA processes are analysed in parallel with the Dolev-Yao attacker [16]. The operations that this attacker model can perform are listed below, but before this we have to introduce new canonical (see Section 1.3.1) names and variables for the attacker. All the canonical names of the attacker are mapped to $n_\bullet$ and all the canonical variables of the attacker are mapped to $z_\bullet$. We also have $\ell_\bullet$ which is a crypto-point in the attacker.

The descriptions of the Dolev-Yao conditions are:

- The attacker initially has the knowledge of the canonical name $n_\bullet$ and all free names of the process P but he can improve his knowledge by eavesdropping on all messages sent on the network.
- The attacker can improve his knowledge by decrypting messages with the keys he already knows. Unless the intended recipient of the message was an attacker, an error (ℓ,$\ell_\bullet$) should be added to the error component ψ which means that something encrypted at ℓ was actually decrypted by the attacker at $\ell_\bullet$.
- The attacker can construct new encryptions using the keys he already knows. If this message is received and decrypted by a principal, then an error ($\ell_\bullet$,ℓ) should be added to the error component ψ which means that something encrypted at the attacker was decrypted by the attacker by a process P at ℓ.

- The attacker can send messages on the network using his knowledge and thus forge new communications.

These conditions enable the attacker to establish scenarios including eavesdropping, modification, man-in-the-middle and replay attacks. The soundness of the Dolev-Yao condition is proved in [8].

As shown in Fig. 1.1, the LySA model of a protocol is analysed in parallel with the attacker model and processed by the LySA-tool (see Section 1.4.4) which implements the static analysis. The results of the analysis are used to validate destination/origin authentication and confidentiality properties of the protocols. If no violation is detected, namely the error component ψ is empty, then it is guaranteed that the protocol satisfies the destination/origin authentication properties. Furthermore, the potential values that are learned by the attacker help us in validating the confidentiality properties. The details as well as the proof of the soundness of the analysis are presented in [7].

Example 4 *In Example 3, we analysed Example 2 in an attack-free setting. Now we add the attacker model and get the following results in addition to the results in Example 3. Since the attacker is able to learn everything sent on the network we have:*

$$K_A, \{K\}_{K_A}^{\ell_A}[\mathsf{dest}\ \ell_B\,] \in \rho(z_\bullet)$$

Therefore, the attacker can decrypt the encrypted part of the message which leads to the violation:

$$(\ell_A, \ell_\bullet) \in \psi$$

Thus we conclude that the encryption at crypto-point ℓ_A which was intended to be decrypted at ℓ_B can be decrypted by the attacker and hence the example protocol is flawed.

1.4 Application on ZigBee Wireless Sensor networks

In this section, we present an application of the analysis method that we explained up to now [17]. This application has many features that make it interesting. First of all, it pinpoints an undiscovered and non-trivial flaw in a

real cryptographic security protocol. Another key issue is that the protocol is being used in one of the latest wireless sensor network standards, ZigBee, that is promising and emerging in the sensor networks field. Therefore, the protocol includes secure components that are known to be secure when they are individually used and some of them are industry standards such as SKKE that we will explain in more details. Still we show that combining proven to be secure components is not sufficient for guaranteeing security properties. Last feature of this application is that we not only use protocol analysis to discover flaws but also to verify our fix proposals.

1.4.1 ZigBee-2007 End-to-End Application Key Establishment Protocol

ZigBee is a fairly new but promising Wireless Personal Area network (WPAN) standard for wireless sensor networks that have very low resource requirements. In parallel with this, the devices that operate in ZigBee networks have limited resources in terms of memory, processor, storage, power, etc. Therefore implementing the security guarantees is a great challenge and the verification of the security properties is of paramount importance.

We start by presenting the key points that are necessary for a clear understanding of the development, and we omit all the details which are not directly relevant to this study. However, a detailed survey on ZigBee security can be suggested as [18] and surely the ultimate source is the ZigBee documentation [20, 21, 22, 23, 19] which is a rather difficult read with hundreds of pages including references to several other standards.

End-to-End Application Key Establishment is the protocol to be used when establishing a Link Key (**LK**) between two ZigBee devices, which are running in *High Security Mode* (which was called *Commercial Mode* in the previous standard, ZigBee-2006 [24]). We will call the devices as initiator (**A**) and responder (**B**). Note that there is also a Trust Center (**TC**), which shares a pairwise secret key with each principal in the network. TC is actually an application that runs on a preferably more powerful ZigBee device referred to as *ZigBee Coordinator* which is unique in the network; whereas the remaining devices might be of type *ZigBee Router* or *ZigBee End Device*, as shown in Fig. 1.3. For a better understanding we should mention that for two ZigBee devices to establish a secure communication, they must share a symmetric key (LK) which they either *receive* from a trusted server (TC) or *create mutually* using a temporary key received from the trusted server.

The scenarios of End-to-End Application Key Establishment are visualized in

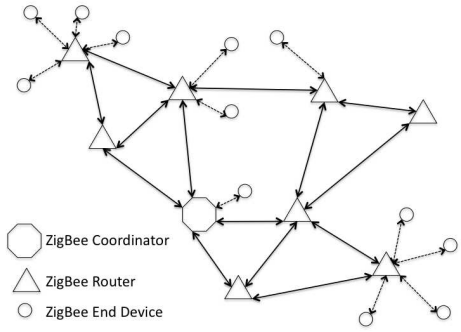

Figure 1.3: ZigBee Network Model

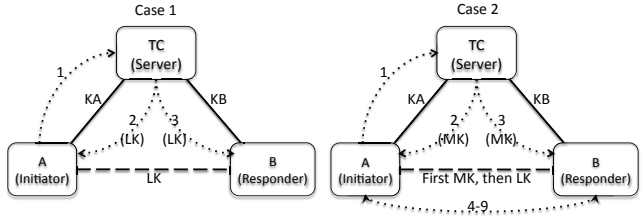

Figure 1.4: ZigBee-2007 End-to-End Application Key Establishment Scenarios

Fig. 1.4. The solid lines represent the already secure communication paths, labeled by corresponding symmetric encryption keys. The dashed lines represent the resulting secure communication paths after a successful protocol run, again labeled by corresponding encryption keys. Finally, the dotted lines are the messages in the protocol labeled by their sequence numbers and the encryption keys they deliver.

ZigBee-2007 End-to-End Application Key Establishment Protocol has two different cases according to the configuration of TC, we will call them as *Case 1* and *Case 2*. In Case 1, TC creates the LK itself and sends it to each principal. Therefore, the initiator and the responder have no role in the creation of the LK. In Case 2, TC creates a temporary shared key called *Master Key* (**MK**) and sends it to each principal. Using this MK, A and B initiate a Symmetric-Key Key Establishment (**SKKE**) procedure to establish an LK. This case allows principals to create an LK *mutually*. SKKE is actually a key agreement scheme employed in the ZigBee End-to-End Application Key Establishment mechanism, and its components are defined in the ANSI X9.63-2001 standard [25]. At the end of (a successful run of) either case, two ZigBee devices will be able to establish secure communication using their pairwise encryption key, LK.

1.4.1.1 Case 1

In Case 1, the initiator begins the procedure of establishing an LK with the responder by sending TC the first message, **request key**, which includes *destination address* (=TC), *requested key type* (=Application Key), and *partner address* (=B). Then TC creates an LK for two principals, and sends it to each principal in two similar **transport key** messages. Since TC is configured to send an LK directly in this case, the *key type* value in the last two messages will be Application Link Key (AppLK). The only difference between these two messages is a boolean value that indicates the initiator (TRUE: message recipient is the initiator, FALSE: message recipient is the responder), and also the principal address'. All the messages in this case are encrypted with the sender/receiver principal's key that is shared with TC (assuming that the security suite is *Encryption-only*). The type of this key can either be Trust Center Link Key (**TCLK**) or Trust Center Master Key (**TCMK**), as defined in the ZigBee specification [20], but for simplicity we will call it *KA* for principal A, and *KB* for principal B. The protocol narration of Case 1 is given in Table 1.7.

Table 1.7: Protocol Narration - Case 1

1. A→TC: {TC, AppKey, B}$_{KA}$
2. TC→A: {A, AppLK, B, TRUE, LK}$_{KA}$
3. TC→B: {B, AppLK, A, FALSE, LK}$_{KB}$

Table 1.8: Protocol Narration - Case 2

1. A→TC: {TC, AppKey, B}$_{KA}$
2. TC→A: {A, AppMK, B, TRUE, MK}$_{KA}$
3. TC→B: {B, AppMK, A, FALSE, MK}$_{KB}$
4. A→B: {B, FALSE, Zero, SKKE}$_{MK}$
5. B→A: {A, TRUE}$_{MK}$
6. A→B: {NA}$_{MK}$
7. B→A: {NB}$_{MK}$
8. A→B: MAC{3,A,B,NA,NB}$_{H(MAC\{A,B,NA,NB\}_{MK},1)}$
9. B→A: MAC{2,B,A,NB,NA}$_{H(MAC\{A,B,NA,NB\}_{MK},1)}$

1.4.1.2 Case 2

In Case 2, the first three messages are almost the same as in Case 1, except in this case TC is configured to send MK, and therefore key type is the Application Master Key (AppMK). The rest of the messages are between the initiator and the responder. In the fourth message, **establish key**, A sends B his request to start SKKE. The values, False and Zero, indicate that there is no *parent* (router, TC, etc.), and no *parent address*, respectively. The fifth message is the response of B to A's SKKE request. Note that these two messages are encrypted by MK, which was received in the previous two messages. The remaining four messages are actually the SKKE protocol itself. Messages 6 and 7 include the *challenges* (NA, NB) of the principals. Messages 8 and 9 are the complex messages which can be computed by both parties to verify each other. A and B create two *message authentication codes* (**MAC**) using their knowledge, besides the MAC key itself is a *hash* (H) of another MAC which they produce using the same knowledge [26]. After the verification, the new LK will be H(MAC{A,B,NA,NB}$_{MK}$, 2), which is a minor variation of the MAC key that was used in the last two messages. The protocol narration of Case 2 is given in Table 1.8.

1.4.2 The Flaw

In wireless networks, it is easy to intercept, forge and inject messages. Without any formal analysis, an experienced eye can see that all the messages in ZigBee-

2007 End-to-End Application Key Establishment Protocol can be replayed when the same long-term encryption keys (KA, KB) are still being used. The reason is the lack of *freshness* elements like nonces, timestamps, etc. This flaw can lead to serious replay attacks, denial of service (DoS) attacks, etc. Even worse, when an old session key is compromised, an attacker can decrypt all the messages by replaying that old session key. In other words, lack of freshness can cause failures in *authenticity* (in the case that principals accept an old session key from a rogue TC) and *confidentiality* (in the case that principals start using a compromised session key).

As can be seen in the narration of the protocol, no freshness indicator is used in the distribution of either LK (in Case 1) or MK (in Case 2, the first three messages). Therefore, all the messages can be replayed. Replay of a message that includes a key is very critical. An attacker can store a message including a key from a previous run of this protocol, and then send the old message to make principals communicate using this old key. If the old key is compromised, then the attacker will be able to decrypt all the messages between two victim principals.

The significance of the security risk that is caused by this flaw may require more explanation. Indeed, the flaw does not disclose any session key but allow reuse of a former key. Besides, brute force attacks or other types of known cryptographic attacks for obtaining the key do not seem practical for the current specification (i.e. the keys are 128-bits). However, disclosure of a key might still be possible without dealing with cryptography, and reuse of an old session key can cause serious risks. An example scenario is given below:

Scenario 1 *A and B established a link key, and had secure communication with the help of that pairwise key. Than B left the network and disclosed the key, which might be by means of hardware (e.g. local key extraction from the chipset such as connecting a debugger, erasing the chip, then freely reading the contents of RAM), or software (e.g. a bug in the implementation that discloses the key after the session expires or terminates with the natural assumption that a new session key will be used for a future session) defects. If B rejoins the network, and run the key establishment protocol with A (no matter which case or security level is chosen), the disclosed key may be replayed by the attacker who can decrypt all the communication using the disclosed key.*

In the ZigBee Specification, the notion of *frame counter* is emphasized as the freshness protection. This approach is not a strong one for several reasons. First of all, a frame counter uses incrementing values rather than random values and rejects frames with a smaller counter value. Second, regardless of the length (which is 32-bits in ZigBee) it is easy to cause overflow to frame counters. As indicated in [27], if an adversary forges a frame with the maximum value (*i.e.*

Table 1.9: Attack Scenario - Case 1

1. A→TC: {TC, AppKey, B}$_{KA}$
2. TC→A: {A, AppLK, B, TRUE, LK}$_{KA}$
3. TC→B: {B, AppLK, A, FALSE, LK}$_{KB}$
1'. A→TC: {TC, AppKey, B}$_{KA}$
2'. M(TC)→A: {A, AppLK, B, TRUE, LK}$_{KA}$
3'. M(TC)→B: {B, AppLK, A, FALSE, LK}$_{KB}$

0xFFFFFFFF) any further frame will be rejected. In addition, using counters is not a novel approach, since in such layered architectures lower layers also used similar counters.

1.4.2.1 Flaw in Case 1

The attack scenario for Case 1 is given in Table 1.9. The first run (messages 1 to 3), is an old run which is intercepted by an attacker. Here, it is appropriate to mention that LK is used like a session key and KA/KB are used like master keys. Therefore, KA and KB are possibly the same in two different runs. The second run in the attack scenario (messages 1' to 3') is initiated regularly, but the last two messages are replayed by the attacker using the messages that are captured from the old run. Furthermore, the attacker does not necessarily need to wait for a message like 1' since he can already replay it, too.

1.4.2.2 Flaw in Case 2

The attack for Case 1 is also possible for Case 2, in which MK is sent without any freshness indicator. Even though LK is created mutually by the use of SKKE in Case 2, a compromised old MK that is replayed to principals before SKKE will allow an attacker to create the LK as well. The attack scenario for Case 2 is given in Table 1.10. The first run (messages 1 to 9) is the old run and it is sufficient for an attacker to capture messages 2 and 3. Then the attacker replays these messages in the new run (messages 1' to 9'). Although the nonce's used in SKKE (exchanged in messages 6 and 7) are different, as long as MK is compromised the attacker can decrypt these messages and learn the nonces as well. As a result, the attacker can still compute the new LK which is actually H(MAC{A,B,NA',NB'}$_{MK}$, 2) (see Section 1.4.1). Therefore, we may conclude that the flaw is critical in both cases.

Table 1.10: Attack Scenario - Case 2

1. A→TC: {TC, AppKey, B}$_{KA}$
2. TC→A: {A, AppMK, B, TRUE, MK}$_{KA}$
3. TC→B: {B, AppMK, A, FALSE, MK}$_{KB}$
4. A→B: {B, FALSE, Zero, SKKE}$_{MK}$
5. B→A: {A, TRUE}$_{MK}$
6. A→B: {NA}$_{MK}$
7. B→A: {NB}$_{MK}$
8. A→B: MAC{3,A,B,NA,NB}$_{H(MAC\{A,B,NA,NB\}_{MK},1)}$
9. B→A: MAC{2,B,A,NB,NA}$_{H(MAC\{A,B,NA,NB\}_{MK},1)}$
1'. A→TC: {TC, AppKey, B}$_{KA}$
2'. M(TC)→A: {A, AppMK, B, TRUE, MK}$_{KA}$
3'. M(TC)→B: {B, AppMK, A, FALSE, MK}$_{KB}$
4'. A→B: {B, FALSE, Zero, SKKE}$_{MK}$
5'. B→A: {A, TRUE}$_{MK}$
6'. A→B: {NA'}$_{MK}$
7'. B→A: {NB'}$_{MK}$
8'. A→B: MAC{3,A,B,NA',NB'}$_{H(MAC\{A,B,NA',NB'\}_{MK},1)}$
9'. B→A: MAC{2,B,A,NB',NA'}$_{H(MAC\{A,B,NA',NB'\}_{MK},1)}$

1.4.3 Proposed Fixed Protocols

We propose fixed protocols that use nonces to ensure freshness of the messages and at the same time the keys. We make use of the vital principles defined on [28]. The narrations of our proposed solution are given in Table 1.11 and Table 1.12 for Case 1 and Case 2, respectively.

In Case 1, we added the nonce of the initiator (**NA**) to the first two messages. This will ensure that when receiving the second message, A will believe that she is communicating with the TC who knows her nonce and also her private key. Note that message 1 can still be replayed but it will be ignored if A does not verify message 2. We inserted two more messages before the last message, so that we use nonces of the TC (**NTC**) and the responder (**NB**) to avoid replay attacks. This will ensure that when receiving the fifth message, B will believe that he is communicating with TC who knows his nonce. Also note that message 3 can still be replayed but the process will be ignored if B does not verify message 5.

Our solution is also applicable to the leaked MK problem in Case 2. Similar to our solution for Case 1, we change the first three messages of Case 2 with five messages that are also given in Table 1.12. Not to confuse with the nonces used in SKKE, the nonces we added are called (**preNA**) and (**preNB**) in Case 2.

Table 1.11: Proposed Fix - Case 1

1. A→TC: {TC, AppKey, B, NA}$_{KA}$
2. TC→A: {A, AppLK, B, TRUE, NA, LK}$_{KA}$
3. TC→B: {B, A, NTC}$_{KB}$
4. B→TC: {TC, A, NTC, NB}$_{KB}$
5. TC→B: {B, AppLK, A, FALSE, NB, LK}$_{KB}$

Table 1.12: Proposed Fix - Case 2

1. A→TC: {TC, AppKey, B, preNA}$_{KA}$
2. TC→A: {A, AppMK, B, TRUE, preNA, MK}$_{KA}$
3. TC→B: {B, A, NTC}$_{KB}$
4. B→TC: {TC, A, NTC, preNB}$_{KB}$
5. TC→B: {B, AppMK, A, FALSE, preNB, MK}$_{KB}$
6. A→B: {B, FALSE, Zero, SKKE}$_{MK}$
7. B→A: {A, TRUE}$_{MK}$
8. A→B: {NA}$_{MK}$
9. B→A: {NB}$_{MK}$
10. A→B: MAC{3,A,B,NA,NB}$_{H(MAC\{A,B,NA,NB\}_{MK},1)}$
11. B→A: MAC{2,B,A,NB,NA}$_{H(MAC\{A,B,NA,NB\}_{MK},1)}$

The fix that we propose is a mechanism that suffices to fix the flaws in the original protocol. There might be other ways to fix, but this is a solution that simply works and has proven (by formal verification) to be secure.

Obviously, the proposed solution would come at a particular cost. Particularly, the number of messages in each protocol is increased by two, and the usage of nonces are required. Transmission of more messages means more power consumption, but for security critical applications (e.g. in Smart Energy, Commercial Building Automation, etc.) this kind of fix which ensures that TC is authenticated to both A and B (i.e. the new LK is not replayed) is necessary, so the additional messages are inevitable. Besides the original protocol in Case 2 already has nine messages (whereas the primitive version, Case 1, only has three), which is a proof that in order to have a sound protocol ZigBee may have longer protocols for the same purpose. The usage of nonces is not a new cost since it is already in SKKE which is employed by Case 2. However, the freshness is preserved for only SKKE but not the protocol itself due to the design mistake of the wrapping protocol.

As we mentioned before, the flaw in End-to-End Application Key Establishment protocol may be visible to an experienced eye but to claim that a fix is flawless, verification using formal methods is crucial. *Static analysis with* LySa is one of

the methods that can be used, which has many advantages such as scalability and the guarantee of termination.

1.4.4 Formal Verification Details

Analysing security protocols without any formal verification method is not a reliable way to find flaws, nor to guarantee that there are no flaws. To make our assertions and arguments sound, we use static analysis to analyse protocols. To be finite, this method is computing over-approximations rather than exact answers, and therefore may lead to false positives. However, when the analysis results tell that the protocol is error-free, then it really is. In other words, no simulation or verification is necessary when the protocols successfully passes static analysis.

The base protocols in Section 1.4.1 are modelled using LySA process calculus and analysed using the LySA-tool[1]. The result supports our claims in Section 1.4.2. The base protocols are prone to replay attacks which will cause serious problems in the case of a leaking key.

The proposed protocols in Section 1.4.3 are also modelled analysed in the same way with the base protocol. The result is successful, namely the proposed protocols do not have any flaws.

The settings that we use to implement the LySA model and verify in the LySA-tool are listed below:

- we check for the origin and destination addresses in each message (by adding them as prefixes such as in IPv4 or IPv6)
- we have the necessary annotations for the encryptions and decryptions
- we allow legitimate attackers in addition to the illegitimate attackers (by adding appropriate zero indices, namely attacker also shares master key with TC)
- we model three groups of (infinite) principals so that we can model man-in-the-middle attacks
- we add an extra message that is encrypted using the session key (to see whether the compromised key can be used)

[1] http://www.imm.dtu.dk/English/Research/Language-Based_Technology/Research/LySa.aspx

- we check all the fields in the messages to have proper values (by pattern matching), except session keys which are newly created (and bound to variables in inputs)

To distinguish between old rounds and new rounds of the protocol we apply a new technique in LySA. We add round indicators to the end of pattern-matched fields in messages and match them in a smart way to distinguish old runs. Using this technique, we can investigate replay attacks successfully.

1.5 Conclusion

Analysing protocols is not a trivial issue, and in this work we presented an analysis method with a detailed application on a new and so called advanced security protocol that uses secure components.

In this approach, we have solid benefits in mainly:

- *solutions always exist and are computed in low polynomial time.* This is an important advantage because approaches based on model checking cannot always guarantee termination, and besides prone to state space explosion problem. Besides the analysis is correct with respect to formal operational semantics, which may be hard to establish in different approaches such as the ones based on modal logic of beliefs (BAN) where the completeness property does not generally hold.

However, those benefits come with a particular cost:

- *lack of trace and counter-example.* Due to the nature of the analysis, there is no trace and no produced counter-example to help flaw discovery. As a result of the over-approximation, false positives may occur and manual inspection is required to match the reported violations to actual flaws.

Another thing we have presented was the usage of protocol analysis in suggesting a secured version of a flawed protocol. Fixing the flaws and proposing secure protocols is another non-trivial job. In this manner, we made use of prudent engineering practices of Gordon and Abadi, and benefited fruitful discussions with Gavin Lowe. One of the points we emphasized was the importance of freshness, and the importance of proper usage of freshness indicators such as nonces, challenges, etc.

We can recapitulate as encryption is not synonymous with security, and its improper use can lead to errors. The proper use should be verified by protocol analysis methods that focus on certain security properties. Along the way in this study, we discovered and documented general guidelines about how to use static analysis for protocol validation. We do believe that such studies are necessary in order to standardise protocols that live up to their stated expectations.

Bibliography

[1] Needham, R.M. and Schroeder, M.D. (1978) Using Encryption for Authentication in Large Networks of Computers. *Comm. of ACM*, vol. 21, no. 12, pp. 993–999.

[2] Burrows, M., Abadi M. and Needham, R. M. (1990) A logic of authentication. *ACM Trans. Comput. Syst.* 8, pp. 18-36.

[3] Lowe, G. (1996) Breaking and fixing the needham-schroeder public-key protocol using fdr. In *TACAS '96*, pp. 147-166.

[4] Lowe, G. (1995) An Attack on the Needham-Schroeder Public-Key Authentication Protocol. *Information Processing Letters*, vol. 56, no. 3, pp. 131–136.

[5] Lowrey, E. S. and Medlock, C. W. (1969) Object code optimization. *Communications of the ACM*, 12(1):13-22.

[6] Busam, V. A. and Englund, D. E. (1969) Optimization of expressions in fortran. *Communications of the ACM*, 12(12):666-674.

[7] Bodei, C., Buchholtz, M., Degano, P., Nielson, F. and Nielson, H.R. (2003) Automatic Validation of Protocol Narration. In *Proc. of CSFW '03*, California, June 30-July 02, pp. 126–140, IEEE, USA.

[8] Bodei, C., Buchholtz, M., Degano, P., Nielson, F. and Nielson, H.R. (2004) Static Validation of Security Protocols. *Journal of Computer Security*, vol. 13, no. 3, pp. 347–390.

[9] Milner, R. (1999) *Communicating and Mobile Systems: The π-calculus.* Cambridge University Press, UK.

[10] Abadi, M. and Gordon, A.D. (1997) A Calculus for Cryptographic Protocols: the Spi Calculus. In *Proc. of CCS '97*, Zurich, 1-4 April, pp. 36–47, ACM, USA.

[11] Nielson, F., Nielson, H.R. and Hankin, C. (1999) *Principles of Program Analysis*. Springer-Verlag, New York.

[12] Nielson, F. and Nielson, H.R. (1997) Flow logic and operational semantics. *Electr. Notes Theor. Comput. Sci.*, 10.

[13] Nielson, H.R. and Nielson, F. (1997) Infinitary control flow analysis: a collecting semantics for closure analysis. In *POPL*, pp. 332-345.

[14] Nielson, H.R. and Nielson, F. (2002) Flow logic: A multi-paradigmatic approach to static analysis. In *T. AE. Mogensen, D. A. Schmidt, and I. H. Sudborough, editors, The Essence of Computation, Lecture Notes in Computer Science*, vol. 2566, pp. 223-244, Springer.

[15] Buchholtz, M., Nielson, H.R. and Nielson, F. (2004) A calculus for control flow analysis of security protocols. *Intl. Journal on Information Security*, vol. 2, no. 3, pp. 145–167.

[16] Dolev, D. and Yao, A.C. (1983) On the Security of Public Key Protocols. *IEEE Trans. on Information Theory*, vol. 29, no. 12, pp. 198–208.

[17] Yüksel, E., Nielson H.R. and Nielson F. (2009) A Secure Key Establishment Protocol for ZigBee Wireless Sensor Networks. *Proc. 24th International Symposium on Computer and Information Sciences (ISCIS 2009)*, pp. 350-355, IEEE.

[18] Yüksel, E., Nielson, H.R. and Nielson, F. (2008) ZigBee-2007 Security Essentials. *Proc. 13th Nordic Workshop on Secure IT-systems*, Copenhagen, 9-10 October, pp. 65–82.

[19] ZigBee-SE-r14 (2008) *ZigBee Smart Energy Profile Specification*, ZigBee Alliance, USA.

[20] ZigBee-2007 (2008) *ZigBee-2007 Specification*, ZigBee Alliance, USA.

[21] ZigBee Stack-r09 (2008) *ZigBee Stack Profile*, ZigBee Alliance, USA.

[22] ZigBee-PRO-r05 (2008) *ZigBee-PRO Stack Profile*, ZigBee Alliance, USA.

[23] ZigBee-2007-HA-r25 (2007) *ZigBee Home Automation Profile Specification*, ZigBee Alliance, USA.

[24] ZigBee-2006 (2006) *ZigBee-2006 Specification*, ZigBee Alliance, USA.

[25] ANSI X9.63:2001 (2001) *Public Key Cryptography for the Financial Services Industry - Key Agreement and Key Transport Using Elliptic Curve Cryptography*. American National Standards Institute, USA.

[26] FIPS Pub 198 (2002) *The Keyed-Hash Message Authentication Code (HMAC), Federal Information Processing Standards Publication 198*. US Dpt of Commerce/N.I.S.T., Springfield, Virginia.

[27] Sastry, N. and Wagner, D. (2004) Security Considerations for IEEE 802.15.4 Networks. *Proc. 3rd ACM Workshop on Wireless Security*, Philadelphia, PA, 01 October, pp. 32–42, ACM, USA.

[28] Abadi, M. and Needham, R. (1996) Prudent Engineering Practice for Cryptographic Protocols. *IEEE Trans on Software Engineering*, vol. 22, no. 1, pp. 6–15.

This figure "fig1.jpg" is available in "jpg" format from:

http://arxiv.org/ps/1205.6678v1

This figure "fig2.jpg" is available in "jpg" format from:

http://arxiv.org/ps/1205.6678v1

This figure "fig3.jpg" is available in "jpg" format from:

http://arxiv.org/ps/1205.6678v1

This figure "fig4.jpg" is available in "jpg" format from:

http://arxiv.org/ps/1205.6678v1